yesyesyesyesCANIMATE.VBX Animation Custom Controlcanimate

Table of Contents

CANIMATE.VBX Version 1.30
Animation Custom Control
©Componere Software, 1994

****
* CANIMATE.VBX - Release 1.00a *
* (c) Componere Software *
* - All Rights Reserved - *
****

On downloading this VBX, you should have received:
CANIMATE.VBX <- Release 1.00a of the animation control library
GAME.EXE <- VBRUN200.DLL required to run
GAME.MAK <- Will work with VB2 or 3
GAME.FRM (also)
BIRD.BMP <- A 3x2 matrix for cell animation
README.TXT (this file)
ORDER.TXT (An order form)

The "game" is a demo of a hero-type player walking back and forth across the stage. A fire is burning in the middle of the stage and every time he walks through it, he hops around a bit. A bird flits about overhead, flying in random patterns to demonstrate varying slopes of travel. The clouds moving overhead are in a control array, update at lesser intervals and move less on each update. The two trees at the bottom demonstrate Zposition; one is in the foreground, one in the background.

Overview

CANIMATE will allow animation controls to be created and manipulated in Visual Basic. In the current release, only 2D animation is supported. There are two types of animation supported: sprite and cell animation. They may be used concurrently or separately. Sprite animation is moving an image about on the screen. Cell animation uses frames, or pictures of the same thing in various states of motion, much like a motion picture. Combining the two techniques results in a much more realistic mode of animation than either technique alone. A bird can beat its wings as it flies about the screen, for example. Sometimes only one is needed; a fire stays in one spot and flickers, while an arrow flies without ever changing its shape.

CANIMATE accomplishes this through the use of two controls: Stage and Player. Stage is the background, a container in which Players are created. Players are animated (or not; they can be simply still pictures) objects to be displayed on the background.

The Stage supplies keyboard and mouse input. It is a control which accepts focus.

The Player is a potentially complex object. In the original design, there were to be several different types of animated objects. But as development proceeded, it became evident that all of them were really just the same object with some properties predefined, and rather than risk not allowing the combination that might be most desirable for a particular situation, a single object was created with all the properties available. Don't be frightened by its complexity; if one is created and a picture assigned, then it will show the picture and not do anything else. If sprite animation is required, the target coordinates are set and moving is enabled. If cell animation is needed, it gets a bit more complicated as described below.

A bitmap assigned to a Player is assumed to be a single picture by default. However, if cell animation is required, the fastest and most efficient way to "package" all the frames is to put them in a single bitmap that is larger than a single frame. The cells are organized in a grid, much like a spreadsheet (only no lines!). On a row, from left to right, are images which should be shown in a logical sequence. If there are multiple logical sequences, then the other sequences are on other rows. For instance, let's say you have a player that is 48 pixels high and 24 pixels wide. On the first row you might have him face to the right, one picture with his feet together, one with them spread apart. On the second row you might have him do the same, only facing left. To do this, you create a bitmap that is 96 pixels wide and 48 pixels wide and draw (or paste - I suggest that somewhere you keep the images unique, but it's up to you, of course) the first row in the upper half and the second row in the second half. Then when this bitmap is assigned to a Player control, set XFrames and YFrames each to 2. Set Cycle to True (I know, I am getting ahead) and he runs in place. But set a target location off to the right, and off he goes. Set the FrameRow to 1 and a target off to the left and he walks that way. The example will help to clear this up a bit.

Events for mouse clicks are generated if Player objects are clicked on. There are additional events that can be generated when a Player object arrives at a target location, runs into another Player object or completes a cycle of cell animation (has shown all the frames on a row). These events must be "turned on"; by default they are not reported, so that Player objects like a flickering fire will have minimal impact on performance.

An important concept to remember is that every object on the Stage is a Player, but that does not mean that you have to move them all with keystrokes. Not at all. Generally speaking, one Player will be your "hero" or "heroine" and the others will be props, hazards, bullets or bad guys.

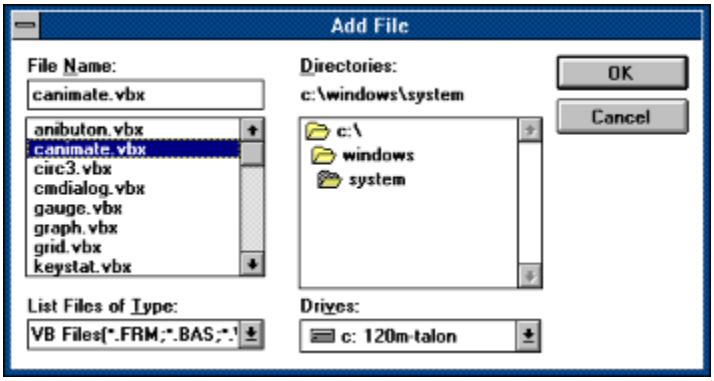
Another important thing to note: There is a lot that can be done with animation besides game programming, such as sprucing up a logo display in the about box or even simulating screens for training.

Planned enhancements will allow 3D scenes to be created, easing the creation of games such as adventures or golf where the player (or ball) moves in a room or landscape. If sound is a common request, it may be implemented, though the multimedia controls should work with CANIMATE without any problem. Perhaps sound might be desirable using the old voice functions for lesser/older machines. Suggestions are welcome.

Quick Start

If the installation did not copy CANIMATE.VBX into your WINDOWS\SYSTEM directory (in some forms CANIMATE is distributed in a ZIP file which you must unzip, then copy the files yourself), then copy it there. If you have a licensed developers release, you will want to copy the development VBX (CANIMATE.DEV) to WINDOWS\SYSTEM\CANIMATE.VBX.

Run Visual Basic to create a new project. From the **File** menu, select **Add File**... and you will see a dialog much like this:



Select CANIMATE.VBX and click on OK.

CANIMATE provides high speed animation through the use of two controls which now appear in your toolbox:

Stage  and Player

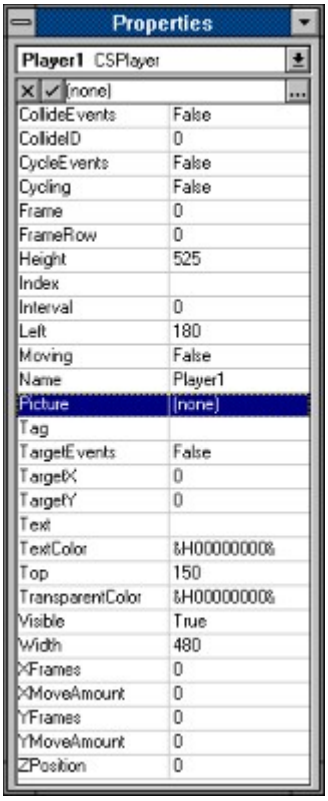
Simply put, you create a stage and then put players on it and give them each their script.

You must create a stage before you can create any players, and all players must be placed on a stage when they are created (they can be moved off stage where they will not be seen during execution, but at design time they must be within the stage).

Click on the Stage  button, then click on the blank form and drag out a fairly good sized rectangle. Press F4 to get the properties window. The first property, BackColor, is selected:

Click the  button and you can select another color (for the purpose of this example, pick something other than white or red), or Picture will allow you to use a bitmap for the background. There is one very important property which you need to learn about right away, though - Interval. Select Interval and change it to 100. This is the number of milliseconds between updates of the Stage and it determines how fast animation can take place. Setting it at 100 instructs the control library to update this Stage 10 times per second.

Now go back to the toolbar and click on the Player  button, then click and drag out a small rectangle (about the size of an icon) near the upper left corner of the Stage control. The Properties window will now have all the properties for a Player:



The Properties window contains a lot more information for a Player than for a Stage, but dont let the number of properties scare you; they are not all needed for each Player and default to reasonable values.

Set the Picture property to the BIRD.BMP file included in the example. To create your own bitmap see the section Creating Animation Bitmaps. The bird in this bitmap is drawn in six different positions:

This is a 3x2 frame bitmap. Each individual frame is one bird in a different state of flight, the top row flying to the right and the bottom flying to the left. In order to let CANIMATE know the layout, set the XFrames property to 3 and the YFrames property to 2.

Since there are multiple frames on each row, we want the control to cycle between them, so set Cycling to True.

Now we must tell it how often to change between frames by setting the Interval property. The Interval property for Player is different than for Stage; it is a multiplier for the Stage Interval. If we set it to 3, then every third time the Stage is updated, the bird will be updated. Set it to 1 so that it is updated each time.

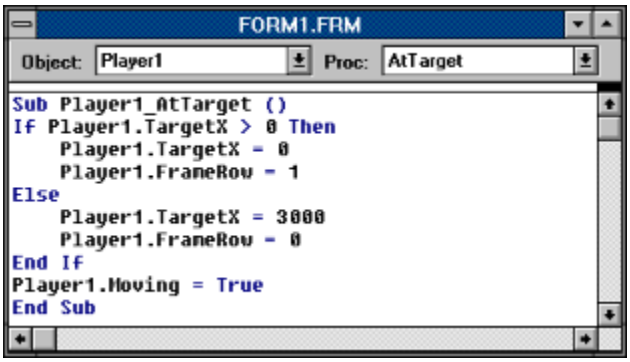
Now press F5 to run. If you have followed all the instructions carefully, the bird will begin beating its wings, and we havent even written any code yet!

But while we have animated the bird, there are still a couple of things we could do to improve it. There is a white rectangle all around the bird, and it isnt flying anywhere, just sitting still. Lets fix that. End execution and go back to design mode.

Change the TransparentColor property to white, using the white block in the upper left hand corner of the color selector. Now the Stage background appears through all of the white areas of the birds bitmap.

Movement is a little trickier. Lets select a target location for the bird to fly to. Try setting the TargetX property to 3000. Set the Moving property to True. Hit F5 to run again. Now the bird flies across the screen, and we still havent written a line of code! Its high time we did. End execution and go back into design mode.

Change the TargetEvents property to True. This will cause the control to generate events whenever the bird arrives at a specified target. All the events have flags which turn them on so that no extra time is taken to generate events for Players which do not have code (like the fire in the sample). Double click on the bird to open the code window. Then select AtTarget from the pulldown list labeled Proc: and enter the following:



Press F5 to run. Now the bird cycles endlessly back and forth until you stop execution. Notice that when we set the target location back to the left, we selected FrameRow 1. Frames are numbered starting with 0, so FrameRow 1 is the second row, which has the bird flying to the left.

Now you should be ready to open the example form and better understand its functions, or build your own. This short tutorial only glossed over the surface of functionality; collision and cycle events from the Player and keyboard events from the Stage provide many more possibilities. Read through Controls, properties and events and A Few Words About Architecture and you will have a much better understanding of the animation controls.

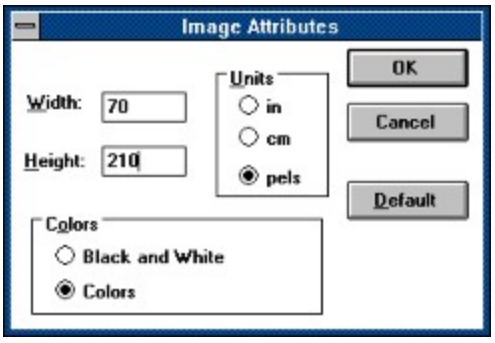
Creating Animation Bitmaps

The bitmaps used by CANIMATE are standard Windows BMP files. One of the easiest way to create them is with the Paintbrush program included with Windows. First, you need to decide how large the image will be. For instance, you might decide to create a person walking that is 32 pixels wide by 64 pixels high.

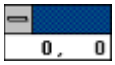
In order to figure the total size for the bitmap, next decide how many different activities you want to be able to animate. To continue using our person walking as an example, you might want walking to the left, walking to the right, and standing still. That will require three rows. How many frames you put on each row will determine how fluid the animation is, though with something as simple as walking, you can get away with just two - one with the legs together, one apart.

Now we know that our total size can be computed by multiplying 32x2 to get the width, and 64x3 to get the height. But before we do that, we might want to round the numbers up to 35 and 70. This will give us a little more room, and the extra space can be painted with transparent colors. It will also make our math (a little later) easier.

Under the **Options** menu, select **Image Attributes** and you will see the following dialog:



Change the **Units** to **pixels** and set the **Width** and **Height** according to the calculation. When you click **OK**, you will be editing a small bitmap for the images.

Under the View menu, select Cursor Position and a small window 

will appear to indicate the position the mouse occupies when in the picture. Go to the color tray and pick a solid color which will not be used in the image. This will be used as the TransparentColor when the bitmap is used in a Player.

Now select the line tool: 

Checking the cursor position window, draw a line from 0,69 to 69,69. If you have trouble with the mouse, you can use the arrow keys to fine tune the cursor position as you move. Draw another line 70 pixels down then bisect them with a vertical line down the middle (34,0-34,209) and you will have a grid of six rectangles each of which should contain one image. After drawing your images (a tip - use the **Pick** menu selection **Flip Horizontal** to turn walking left into walking right), simply fill in all the left over space with the transparent color used to draw the lines using the spill bucket tool.

There are many better programs for serious bitmap editing, but the Paintbrush program will produce adequate bitmaps for use in animation. There are many efficiencies which can be learned by opening a second larger bitmap

and cutting and pasting back and forth, but this section is not intended to teach everything you ever wanted to know about bitmap editing; it will get you started so you can learn by doing.

Ordering Information

CANIMATE is a licensed product of Componere Software. The unregistered evaluation version may be used to develop for any length of time you wish; however, you cannot ship it with a product, even if that that product is free.

The evaluation version may be freely redistributed as long as all the files accompany it, and it is not distributed as part of another product. A note to BBS operators: A window will appear in the background indicating that this is an unregistered version. This is to be expected and it is an indication that it is in fact the evaluation version. There is tamper detection logic included which requires that the evaluation copyright window not be disabled.

You may purchase a registered version from:

Componere Software
1381 Kildaire Farm Rd #119
Cary, NC, 27511-5525

Credit Card Orders are available from PsL.

There are two prices for CANIMATE.VBX. If you want to redistribute applications which use the animation provided, it is $39 (US dollars only) for unlimited redistribution rights of the runtime version, plus a single use license for the development version. If you only want the development version for personal use, with no redistribution rights, you can order it for $19. The advantage of the development version over the provided demo version is that you will be notified of updates, given support and you will be able to sleep at night :)

An Order Form is included in this help file. Please use it to order.

Only checks or money orders for US dollar amounts can be accepted at this time. Please send email if you need to arrange some other form of payment.

If you have any questions or comments, contact:
Email: Componere@aol.com

If CANIMATE does not meet your animation needs, please drop me a note; I may address your concern in a future version, and there may be a newer version already available.

Credit Card Orders

CREDIT CARD ORDERS ONLY -

You can order with MC, Visa, Amex, or Discover from Public (software) Library by calling 800-2424-PsL or 713-524-6394 or by FAX to 713-524-6398 or by CIS Email to 71355,470. You can also mail credit card orders to PsL at P.O.Box 35705, Houston, TX 77235-5705.

THE ABOVE NUMBERS ARE FOR ORDERS ONLY.

Ask for productt #11423 - CANIMATE.VBX
Prices are: $25 US, $26 Canada and $28 overseas for the single user version.
Prices for redistributable version are $45 US, $46 Canada and $48 overseas.
Prices include shipping.

Any questions about the status of the shipment of the order, refunds, registration options, product details, technical support, volume discounts, dealer pricing, site licenses, etc, must be directed to Componere Software.

To insure that you get the latest version, PsL will notify us the day of your order and we will ship the product directly to you.

Order Form

Select File=>Print Topic from menu to print this form.

Componere Software
1381 Kildaire Farm Rd #119
Cary, NC, 27511-5525

Quantity	Item	Price Each	Total Price
___	Redistributable CANIMATE.VBX Version 1.3 with unlimited run time and single developer license.	39.00	_____
___	Single developer license with no redistributable run time version.	19.00	_____
		SubTotal	_____
		NC residents add 6.5% Tax	_____
		Total	_____

Make check payable to Componere Software
Prices are in US dollars

Shipping Address: _____

_____

_____

_____

Phone: _____

Email: _____

Questions?
Contact Componere@aol.com by email

Controls, properties and events

Stage 

Overview
Properties
Events

Player 

Overview
Properties:
Events:

Stage Overview

Stages are windowed controls on which Players are displayed. A Stage only has a few properties and is primarily used as a backdrop. A background color or bitmap can be selected for display. The Stage also provides the timing for all actions within it, using the Interval property.

A Stage should be created before any Players and should not be destroyed while any Players are still present within it.

In general, it is a good idea to use only a single Stage. The Stage controls consume a large amount of resources.

Stages are required to do any animation in CANIMATE.

Stages accept focus and should be used for input in games. The KeyDown event in particular was designed for game input as it ignores repeat characters. The idea behind this is to use directional keys to aim off in the distance when pressed, then stop a players motion when they are released. Trying to react to each repeated key will consume far too much time and slow your game. See the included example program.

Stage Properties

Standard Properties:

- BackColor
- BorderStyle
- Height
- Index
- Left
- Name
- Tag
- Top
- Visible
- Width

Custom Properties:

- Interval
- Picture

Stage Interval Property

Interval - This is the number of milliseconds to set timer events for animating objects. Also see interval for Player control. Interval for the Stage control determines the at which Players can be animated.

Stage Picture Property

An optional bitmap. A stage may either use a bitmap or background color.

Stage Events

Click

KeyDown

KeyUp

KeyChar

Stage Click Event

Sub Stage1_Click ()
Standard mouse click event

Stage KeyDown Event

Sub Stage1_KeyDown (WP As Integer)
A key was pressed.
A WM_KEYDOWN message was received. WP is the wParam or virtual key code.

NOTE: There is one major difference between this event and the Windows WM_KEYDOWN. Repeat keys are not reported (they are by the KeyChar event). This works much better in games, allowing you to set a target far away on the down, then stop moving on the up without getting the intervening repeat characters. Trying to respond to each key in repeat mode in an arcade game takes too long.

Stage KeyUp Event

Sub Stage1_KeyUp (WP As Integer)
A key was released. WM_KEYUP message was received. WP is the wParam or virtual key code.

Stage KeyChar Event

Sub Stage1_KeyChar (WP As Integer)
A keystroke was translated into a character.
A WM_CHAR message was received. WP is the character as an int.

Player Overview

Players are the heart of CANIMATE. They are the actual animated objects. Players must be created on a Stage at design time, but they may be moved to negative coordinates or large positive coordinates to move off Stage at run time. However, even when moved off a Stage, the link between a Player and the Stage on which it was created is permanent.

Players are generally assigned bitmaps which may contain one or more images in a grid layout. If a grid is used, the cells within each row are treated as a sequence when Cycling is enabled for cell animation.

Players have an invisible color so that they need not be rectangular. This transparency is used not only for drawing, but also for collision detection.

Players can generate events back to Visual Basic if the event flags are enabled. Flags are used so that time is not wasted by generating events which are not needed.

Players can be animated or inanimate objects and may or may not have Visual Basic code associated with them. As demonstrated in the tutorial, a Player can be animated without writing any code whatsoever.

Player Properties

NOTE:
Height and Width are a little different with a Player than with most other controls. Frames are not stretched to fill a larger rectangle; the extra space is ignored. If the space is too small, it will not clip the frame being displayed, nor shrink it. So the size of a Frame (or Picture if XFrames and YFrames are both 1 or less) really determines control size. All position and size properties are in twips. ScaleMode doesn't work since Players are inside of the Stage object, and there doesn't seem to be any clean workaround.

Standard Properties:

- Height
- Index
- Left
- Name
- Tag
- Top
- Visible
- Width

Custom Properties:

- CollideEvents
- CollideID
- CycleEvents
- Cycling
- Frame
- FrameRow
- Interval
- Moving
- Picture
- TargetEvents
- TargetX
- TargetY
- TransparentColor
- Text
- TextColor
- XFrames
- XMoveAmount
- YFrames
- YMoveAmount
- ZPosition

Player CollideEvents Property

If set to True, events will be generated if the Player collides with another visible Player control which has a non zero CollideID. See Collide

Player CollideID Property

This is used to identify the Player or type of Player in a collision. It does not have to be unique. For instance, you might set all hazards to 1 and walls to 2 in a game. Then if you get a Collideevent with a 1, you can "kill" your player, while simply stopping for a 2.

Player CycleEvents Property

If set to True, events will be generated each time all the frames on a row have been shown. See Cycle

Player Cycling Property

If True, the Player will cycle through frames on a row. This enables cell animation.

Player Frame Property

This is a number from 0 to XFrames-1 which indicates the frame currently being displayed from the current FrameRow.

Player FrameRow Property

This is a number from 0 to YFrames-1 which indicates which row of frames are currently being displayed.

Player Interval Property

This indicates how often the Player should be updated. Each time an update occurs, the Player will be moved if it is moving toward a target and/or have the next frame shown if cell animation is being used. This is a multiplier for the Stage's Interval. If the Interval of the Stage is set to 111 and the Interval of the Player is set to 3, it will be updated every 333 milliseconds (approx.) or about 3 times a second, for example.

Player Moving Property

If set to True, the Player will begin moving toward the target location if/when it is different than the current location. This is also a state flag to indicate whether or not a Player is moving, and will be set to False whenever a Player reaches the target specified by the TargetX and TargetY values, which must be set before changing Moving to True.

Player Picture Property

The bitmap. If XFrames and YFrames are 1 or 0, then it is simply a single picture. Otherwise, it is a collection of frames all of equal size arranged in a grid.

Player TargetEvents Property

If set to True, events will be generated when the Player reaches a target location. See AtTarget

Player TargetX Property

The target location X coordinate in twips. A target location must be set prior to setting Moving to True to enable motion.

Player TargetY Property

The target location Y coordinate in twips. A target location must be set prior to setting Moving to True to enable motion.

Player TransparentColor Property

The color in the bitmap Picture which should not paint over objects or background beneath. Also, a collision does not occur if either of the overlapping areas is completely transparent.

Player Text Property

A character string to write into the control.

Player TextColor Property

The color to use for Text

Player XFrames Property

The number of frames on each row of the Picture if > 1.

Player XMoveAmount Property

The maximum number of twips to move on each update in the X direction when moving toward a target. The Player may move less in some cases as dictated by slope of the line of travel or proximity to the target location. A reasonable default value is used if it is not set.

Player YFrames Property

The number of rows of frames in the bitmap Picture if > 1

Player YMoveAmount Property

The maximum number of twips to move on each update in the Y direction when moving toward a target. The Player may move less in some cases as dictated by slope of the line of travel or proximity to the target location. A reasonable default value is used if it is not set.

Player ZPosition Property

Determines paint order; the lower the number the later it is painted (0 = topmost). If this is not set, the order will be the order in which the controls are created. Collisions may still be reported even if ZOrders are not equal. You might choose to set the ZOrder of some Player control, like a bush, lower than a Player that is your "hero" so that the hero can hide behind the bush (some hero :)

Player Events

AtTarget

Click

Collide

Cycle

Player AtTarget Event

Sub Player1_AtTarget ()
The player was in motion and has arrived at the target location. TargetEvents and Moving must be True for this event to be generated. Moving will be set to False before this event is triggered. The Intervals for both Player and Stage must also be non zero for this event to ever occur.

Player Click Event

Sub Player1_Click ()
The standard mouse click event.
This Event can be a little tricky with Players, due to their visual movement.

Player Collide Event

Sub Player1_Collide (ID As Integer)
The Player is in motion and has collided with another Player control which has a non zero CollideID. The ID is the CollideID of the other Player control. CollideEvents must be True for this event to be generated. The Intervals for both Player and Stage must also be non zero for this event to ever occur.

Player Cycle Event

Sub Player1_Cycle ()
The Player is cycling through frame animation and has completed a cycle (shown all the images on a frame row). If nothing is done to change the state, the Player will continue cycling, starting at Frame 0 on the current FrameRow. CycleEvents and Cycling must be True for this event to be generated. The Intervals for both Player and Stage must also be non zero for this event to ever occur.

Trouble Shooting

Nothing moves or cycles through frames

Don't forget to set the Interval values for Stage and Player!!!!

Don't forget to set the Interval values for Stage and Player!!!!

If Interval is zero, it is assumed that you want to freeze frame; in fact, that is the best way to implement a pause in your game.

Player doesn't move or stops moving

To get a player to move, set TargetX, TargetY and set Moving to True. When the player arrives at the target, it will stop and Moving will be set to False; you have to retarget and set Moving true again to get more movement.

Player never cycles through frames

Cycling must be set to True to use cell animation. Cycling will remain True after a Cycle event (unlike Moving) unless you turn it off.

No events generated for a Player

You must set the flag for the events you want (CycleEvents, CollideEvents, TargetEvents) or you will not get any events. One of the ways the speed is maintained is by not calling the Visual Basic handler unless it is requested. Also, Interval must be non zero.

No collision events being returned for a player

The CollideEvents must be True and the CollideID of the Player objects which it runs into must not be zero.

Colors get confused when multiple bitmaps are up

All the bitmaps must share the same palette. If you use the Windows Paintbrush program, they will all use the system palette.

Movement is "odd"

Make sure that your values for the grid are right, and that the frames are evenly spread. Check the movement amounts and remember that they must be in twips. Make sure you don't do too much processing at events.

Colors seem to mix while moving

Some colors are composite colors; if you zoom in on them in paint, you will see alternating bits of different colors which appear as a single color when zoomed out. If you pick one of these as your TransparentColor, then these bits of color will allow the background to come through. You need to make sure that the color you pick to be transparent is not used at all in the foreground of the picture; not even "mixed" in another color. A tip - you can get some interesting effects by doing this on purpose, for ghosting or flames.

Events don't come in while a message box is up

That's the way Visual Basic works. Events are thrown away when a message box comes up until it goes away. You should set the Stage Interval to zero while it is up to freeze animation, unless missing events is unimportant.

Animation stops while debugging

Only while in break mode. This is on purpose. You would find it impossible to debug if events continued to happen (they get ignored by Visual Basic) while in break. This also allows you to examine the "state of the world" during break at your own pace. Each time you Go, it runs full speed.

Not all events delivered when colliding with multiple players

If you destroy (unload) any Stage or Player as a result of any event, no more events will be generated until the next Interval. This is a safety feature. See the "A Few Words About Architecture" section for further information.

Strange problems may occur with very large bitmaps

To maintain speed, everything is kept in memory. If you have very large bitmaps, a lot of memory is used. CANIMATE may be described as a "resource hog". The initial design called for small bitmaps, but no limit is enforced. This is sort of a "your mileage may vary" disclaimer. Too many small bitmaps are just as bad or worse than a few big ones.

Cannot select some players during design time

You will have to use the drop down in the property list to get at them. This is a problem further explained in the "A Few Words About Architecture" section below. The Players are "under" the Stage. This happens with cutting and pasting and double clicking.

Clipcontrols set to false causes problems with drawing

Set it to True! More architecture concerns.

Unloaded controls sometimes do not disappear

Hide controls before unloading if this happens. In order to keep the speed of animation high, some Visual Basic instructions to draw controls are ignored (it asks far too often). Unfortunately this logic occasionally causes a draw to be missed when controls are destroyed.

Getting More Help

If you run into a problem which is not documented, and you suspect may be a bug, or if you have other questions or comments on the CANIMATE.VBX product, you may contact Componere Software by email or FAX. Email is preferred and will result in a more prompt response.

Internet Email Address: Componere@aol.com

FAX: (919) 481-0919

Be sure to put **Componere Tech Support** in the title of the message. Always include the version number and describe the problem as explicitly as possible, including any code examples. If your problem concerns a particular bitmap image, be specific as to its size, number of colors (the format, not necessarily the number actually used; a 16 color bitmap of a black and white image is still a 16 color bitmap) and the type of monitor on which it is being displayed.

Distributing Your Application

Only licensed run time copies of CANIMATE.VBX may be redistributed with your application, and only if you have licensed the right to redistribute (see Ordering Information) CANIMATE.VBX. Neither the demonstration version nor the developers version may be redistribute with your application.

If you have a licensed copy which includes the run time redistributable version, you may redistribute only the run time version. There are no royalties for redistribution of the run time version.

If you used the defaults when running the Install program, your redistributable version will be C:\CANIMATE\REDIST\CANIMATE.VBX. After creating your EXE file from your Visual Basic program, copy this VBX into C:\CANIMATE (or wherever you keep CANIMATE.VBX while developing) and run your program to ensure that it is working properly. Create your installation disks using this version of CANIMATE.VBX. It is smaller, and does not display any license information. It is a run time only version, and does not support development.

To switch back to development mode, simply copy the CANIMATE.VBX from C:\CANIMATE\DEVEL over the run time version.

A Few Words About Architecture

Visual Basic Custom Controls (VBXs) are designed to be stand alone components. This means that there should not be interdependence between various controls on the screen. They may require a Form on which they are placed, but little else.

CANIMATE would not work if it strictly adhered to the model. You cannot detect collisions or negotiate painting to eliminate flicker without knowledge of where other controls are located.

A Stage control is a window on which Players are painted, and this window also runs a timer for generating Intervals to update all the Players. In order to do this, the Stage must be the "Parent" of all Players. Sometimes Players will be created in a manner such that Visual Basic does not consider it to be located in a Stage. The Stage and Player still have an internal link between them, as there is no way to force Visual Basic to make the connection, and too much functionality would be lost if these methods (especially Loading) were not allowed. This does cause a design time interface problem, however; you will not be able to click on such objects because Visual Basic has placed them "under" the Stage.

When controls are being updated, the Stage control is running through a list of controls, updating each as necessary. The update may result in an event, such as a Collision. If the VB code run for the event causes a control to be destroyed, the list that the Stage is using may be corrupted (in fact, the Stage may itself be destroyed), so for safety's sake, no further updates are made until the next Interval.

The Intervals are generated by the "standard" windows timer events. Even though these can be requested as milliseconds, the true granularity is not nearly so fine. It is around 50 milliseconds.

Since the Stage processes timer events which may cause events to be generated to Players, it is quite possible that one event may be generated while another is still being processed. VB does not allow this. It does not react violently or cause an error, but the event is lost. Note that timer events only happen if Interval is non zero and VB is accepting input. If your event handlers for Player and Stage do not bring up other forms or message boxes, this will not be a problem. Setting the Interval to zero will stop the timer, and setting it back to its previous value will start it back up.

CANIMATE was described earlier as a resource hog. This is very true. Speed is the first consideration, and keeping everything in memory is standard procedure. Animation is typically a foreground activity, so resource sharing was not a design goal. If you need to share, you should limit the size of the Stage and number of Players, or unload the Stage or Form containing it.

There is a special property available at run time only for both Stage and Player. This property is BitMap and can be set to a bitmap handle obtained by calling the LoadBitmap API. Setting this property will supersede the Picture setting. It is up to the programmer to also clear this property before deleting the bitmap. This property was provided as an option for advanced developers to be able to quickly change bitmaps from a DLL. It is not necessary for most applications. It is not documented in the standard properties section as misuse of this property, intentional or otherwise, may result in GP faults or other problems. It should be used with extreme care, if at all.

This section was added in hopes that understanding a little bit about the way that the VBX works might help you make decisions about the design in your program to alleviate possible problems. It is not a disclaimer - bugs will certainly be addressed, but they must be addressed within the limitations imposed on VBXs.